



Using the NicheStack TCP/IP Stack - Nios II Edition

Tutorial



101 Innovation Drive
San Jose, CA 95134
www.altera.com

TU-01001-3.0



[Subscribe](#)

Copyright © 2011 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, and specific device designations are trademarks and/or service marks of Altera Corporation in the U.S. and other countries. All other words and logos identified as trademarks and/or service marks are the property of Altera Corporation or their respective owners. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



Chapter 1. Using the NicheStack TCP/IP Stack

Introduction	1-1
Hardware and Software Requirements	1-2
Tutorial Files	1-2
Hardware Design Files	1-2
Software Files	1-2
Software Development Flow	1-3
Creating a New Nios II Project	1-4
Configuring the BSP	1-7
Examining the Nios II Simple Socket Server Project Files	1-11
Building and Running the Nios II Simple Socket Server Project	1-11
Interacting with the Nios II Simple Socket Server	1-13
Nios II Simple Socket Server Overview	1-16
Software Naming Conventions	1-16
Software Architecture	1-17
MicroC-OS/II Resources	1-19
Tasks	1-19
Inter-Task Communication Resources	1-20
NicheStack TCP/IP Stack Initialization	1-20
Nios II Simple Socket Server Implementation Details	1-21
Important NicheStack TCP/IP Stack Concepts	1-22
Error Handling	1-22
NicheStack TCP/IP Stack Default Task Creation	1-22
Creating Tasks that Use the NicheStack TCP/IP Stack Sockets Interface	1-23
Task Priorities in the Nios II Simple Socket Server Design	1-25
MicroC/OS-II Internal Tasks	1-25
NicheStack TCP/IP Stack Internal Tasks	1-25
Networking Initialization Task	1-26
User Networking Tasks	1-26
User Non-Networking Tasks	1-26
PHY Monitoring Task	1-26
Task Stack Size	1-26
Where to Go Next	1-27

Appendix A. Hardware Setup Details

Introduction	A-1
Network Connection	A-1

Additional Information

Document Revision History	Info-1
How to Contact Altera	Info-1
Typographic Conventions	Info-1

This tutorial introduces you to the Nios® II Software Build Tools (SBT) for Eclipse™ using the MicroC/OS-II and NicheStack TCP/IP Stack development flow. It shows you how to use the Nios II SBT for Eclipse to create a new Nios II project that configures, builds, and runs a MicroC/OS-II and NicheStack TCP/IP Stack program on an Altera® development board.

Introduction

This tutorial familiarizes you with the NicheStack TCP/IP Stack – Nios II Edition (NicheStack TCP/IP Stack) software component. The tutorial covers the following topics:

- Configuring and initializing the NicheStack TCP/IP Stack software component
- Managing a TCP/IP connection with MicroC/OS-II real-time operating system (RTOS) tasks
- Using the Nios II SBT for Eclipse to develop programs with the NicheStack TCP/IP Stack software component

The Nios II SBT for Eclipse offers software designers a rich development platform for Nios II applications. The Nios II SBT for Eclipse contains the MicroC/OS-II RTOS and the NicheStack TCP/IP Stack software component, providing designers with the ability to quickly build networked embedded systems applications for the Nios II processor. This tutorial provides step-by-step instructions for building a simple program based on the MicroC/OS-II RTOS and NicheStack TCP/IP Stack networking stack.

This tutorial describes C software files that demonstrate communication with a telnet client on a development host PC. The telnet client offers a convenient way of issuing commands over a TCP/IP socket to the Ethernet-connected NicheStack TCP/IP Stack running on the Altera development board with a simple TCP/IP socket server example. The socket server example receives commands sent over a TCP/IP connection and turns LEDs on and off according to the commands. The example consists of a socket server task that listens for commands on a TCP/IP port and dispatches those commands to a set of LED management tasks.



The Nios II target system does not implement a full telnet server.



For more information about MicroC/OS-II for the Nios II processor, refer to the *MicroC/OS-II Real-Time Operating System* chapter of the *Nios II Software Developer's Handbook*.



For more information about NicheStack TCP/IP Stack initialization and configuration for the Nios II processor, refer to the *Ethernet and the NicheStack TCP/IP Stack – Nios II Edition* chapter of the *Nios II Software Developer's Handbook*.

Hardware and Software Requirements

This tutorial requires the following hardware and software:

- Quartus® II software version 11.0 or later
- Nios II Embedded Design Suite (EDS) version 11.0 or later
- One of the following Altera development kit boards:
 - Nios II Embedded Evaluation Kit (NEEK), Cyclone® III Edition
 - Embedded Systems Development Kit (ESDK), Cyclone III Edition
 - Stratix® IV GX FPGA Development Kit
- Altera® USB-Blaster™ cable
- RJ-45 connected Ethernet cable on the same network as the PC development host

To complete this tutorial, you must install the Nios II SBT for Eclipse and you must connect your Altera development board to a host PC on the Ethernet and USB/JTAG ports. For hardware setup instructions, refer to [Appendix A, Hardware Setup Details](#).

Tutorial Files

The files for this tutorial are available in two **.zip** files on the Altera website. One file contains the hardware design example and the other contains the software program files.

Hardware Design Files

The [Nios II Ethernet Standard Design Example](#) page of the Altera website contains the hardware design files to use with this tutorial. Navigate to the web page and locate the Nios II Ethernet Standard design example **.zip** file that corresponds to your board. Download and unzip the file in a directory of your choosing. The tutorial uses *<tutorial_files>* to refer to this directory.

Software Files

The [NicheStack tutorial software files .zip](#) file (next to the link to this document on the [Literature: Nios II Processor](#) page of the Altera website) contains the software program files to use with this tutorial. Download and unzip the file in a directory of your choosing. The tutorial uses *<tutorial_files>* to refer to this directory.

You can locate the following software files in the `<tutorial_files>\nichestack_tutorial` directory. These software files constitute the Nios II Simple Socket Server application for this tutorial:

- **alt_2_wire.c**—Contains utilities that provide a low-level interface to the EEPROM devices.
- **alt_2_wire.h**—Defines utilities that provide a low-level interface to the EEPROM devices.
- **alt_eeprom.c**—Contains utilities that read, write, dump, and fill the contents of the EEPROM devices.
- **alt_eeprom.h**—Defines utilities that read, write, dump, and fill the contents of the EEPROM devices.
- **alt_error_handler.c**—Contains three error handlers, one each for the Nios II Simple Socket Server, NicheStack TCP/IP Stack, and MicroC/OS-II.
- **alt_error_handler.h**—Contains definitions and function prototypes for the three software component-specific error handlers.
- **iniche_init.c**—Defines `main()`, which initializes MicroC/OS-II and NicheStack TCP/IP Stack, processes the MAC and IP addresses, contains the PHY management tasks, and defines function prototypes.
- **led.c**—Contains the LED management tasks.
- **niosII_simple_socket_server.c**—Defines the tasks and functions that use the NicheStack TCP/IP Stack sockets interface, and creates all the MicroC/OS-II resources.
- **niosII_simple_socket_server.h**—Defines the task prototypes, task priorities, and other MicroC/OS-II resources used in this tutorial.
- **tse_my_system.c**—Defines the global structure of type "alt_tse_system_info", named "tse_mac_device" which describes the TSE configuration.

Software Development Flow

The process for creating a NicheStack TCP/IP Stack and MicroC-OS/II software image for the Nios II processor consists of the following general steps:

1. Creating a new Nios II SBT for Eclipse C/C++ application project with the simple socket server project template
2. Configuring a board support package (BSP) project, including MicroC/OS-II and the NicheStack TCP/IP Stack software component
3. Building the application project
4. Running and debugging the application project

Creating a New Nios II Project

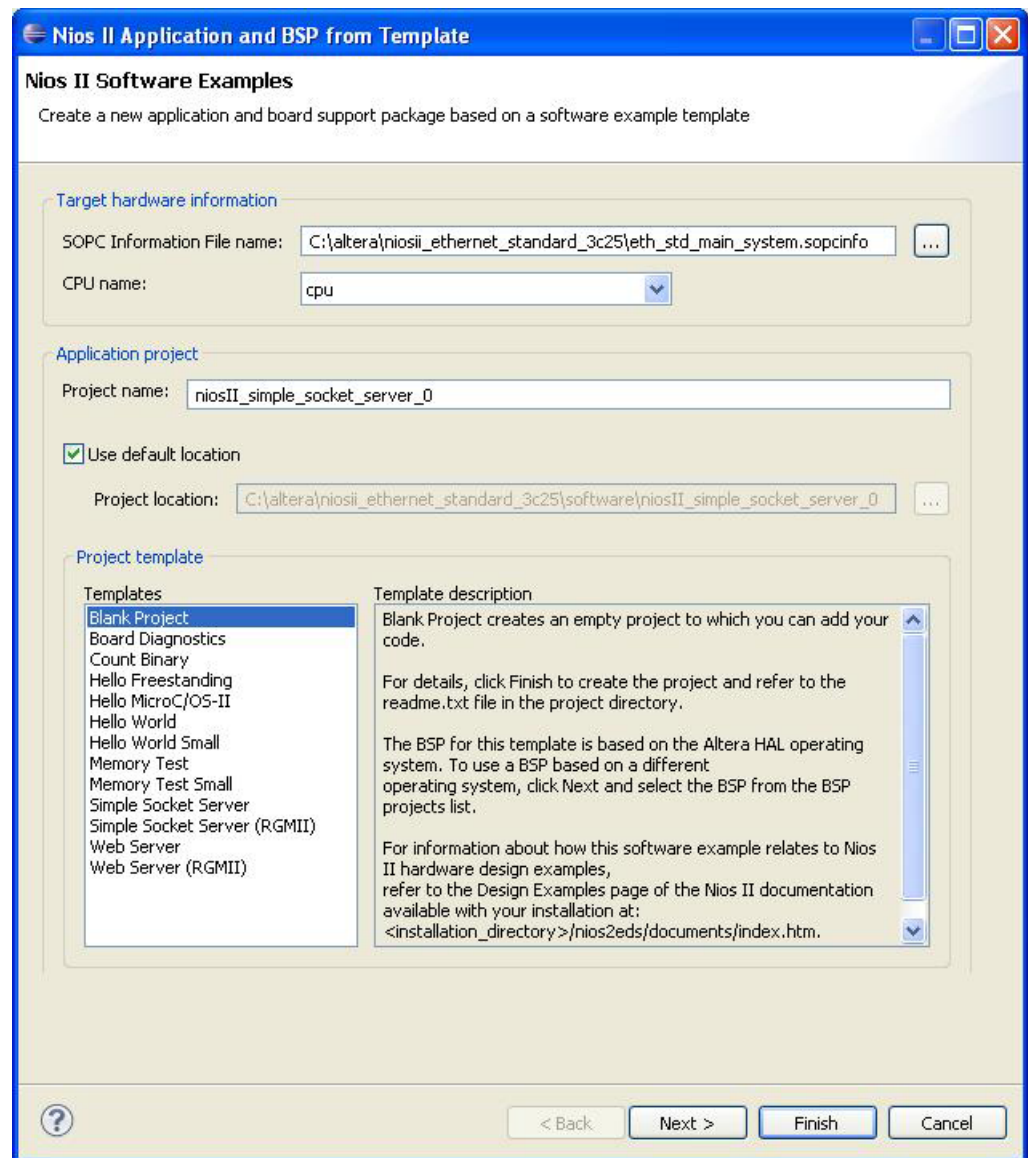
In this section, you create a new Nios II SBT for Eclipse project using a project template. To do so, follow these steps:

1. Start the Nios II SBT for Eclipse by performing one of the following actions:
 - On Windows, on the Start menu, point to **All Programs > Altera > Nios II EDS <version>**, and click **Nios II <version> Software Build Tools for Eclipse**.
 - On Linux, open a Nios II Command Shell and type `eclipse-nios2` ↵.
2. On the File menu, point to **New** and click **Nios II Application and BSP from Template**. The first page of the **Nios II Application and BSP from Template** wizard appears.
3. Under **Target hardware information**, browse to and open the `<tutorial_files>\niosii_ethernet_standard_<board>\eth_std_main_system.sopcinfo` SOPC Information File (`.sopcinfo`). The **SOPC Information File name** box contains the path to the `.sopcinfo` and the **CPU name** box contains the name of one of the available Nios II processors as defined in SOPC Builder. The hardware design of the tutorial contains a single processor, so the software automatically selects the single processor.
4. In the **Project name** box, type `niosII_simple_socket_server_0`. The **Project location** fills in for you automatically.
5. Verify **Use default location** is on.

- Under **Project template**, select **Blank Project**.


Figure 1–1 shows the state of the **Nios II Application and BSP from Template** wizard at this point in the tutorial.

Figure 1–1. Nios II Application and BSP from Template Wizard



- Click **Next**. The second page of the **Nios II Application and BSP from Template** wizard appears.
- Select **Select an existing BSP project from your workspace**.
- Click **Create**. The **Nios II Board Support Package** dialog box appears.
- In the **BSP name** box, type `niosII_simple_socket_server_0_bsp`.
- In the **Operating system** list, select **Micrium MicroC/OS-II**.

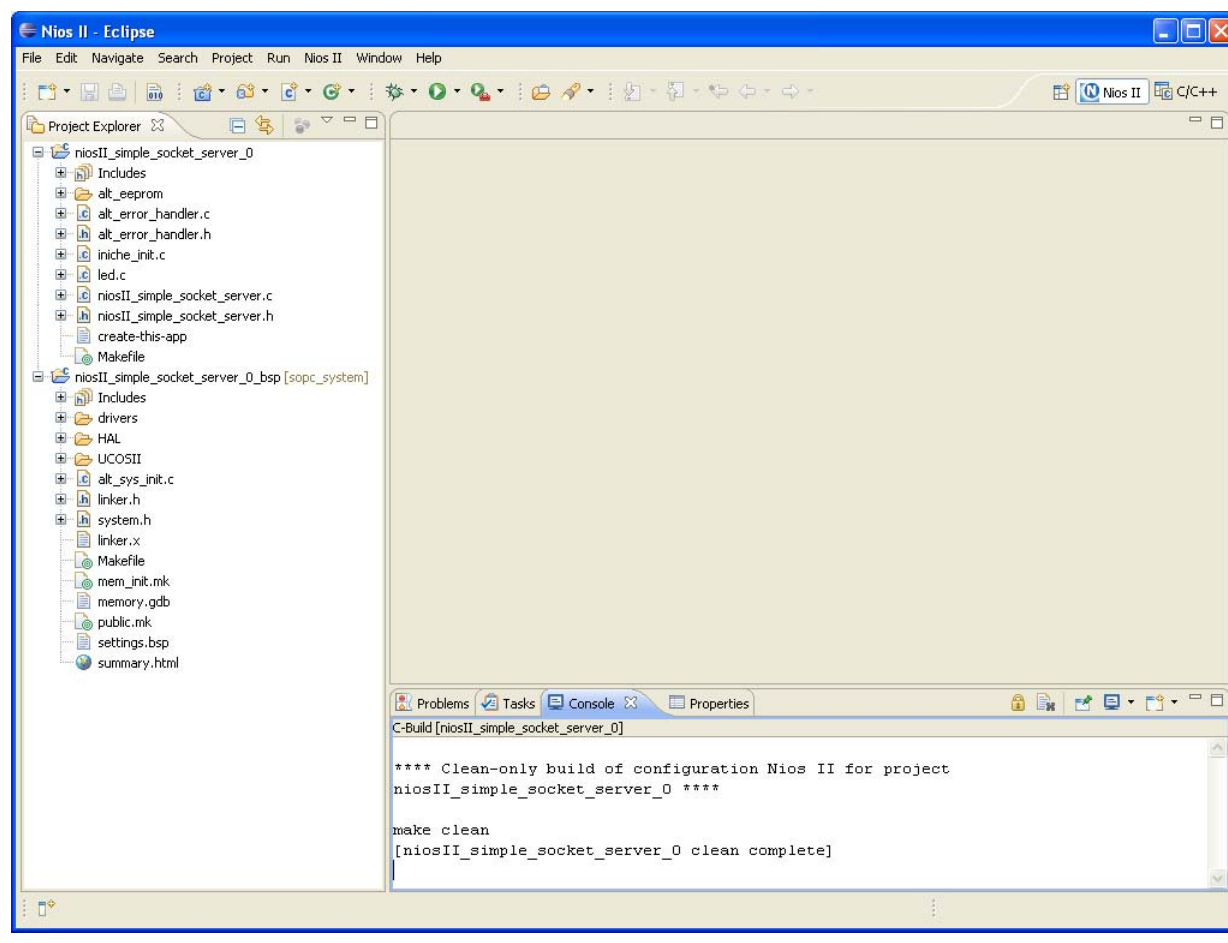
12. Click **Finish**. The wizard creates a BSP project and closes the **Nios II Board Support Package** dialog box.
13. Click **Finish**. The wizard creates an application project.

 If the wizard prompts you to open the Nios II perspective, click **Yes**. If the **Finish** button is grayed out, click **Cancel** to close the previous GUI. Repeat Step 2 - 8, select **niosII_simple_socket_server_0_bsp**, and click **Finish**.

14. With a file management tool (such as Windows Explorer), drag and drop all the Nios II Simple Socket Server source files and folders from the `<tutorial_files>\nichestack_tutorial` folder to the **niosII_simple_socket_server_0** folder in the Nios II SBT for Eclipse Project Explorer view.
15. Select **Copy files and folders** in the **File and Folder Operation** dialog box, and click **OK**.

Figure 1-2 shows the application and BSP projects in the Project Explorer view at this point in the tutorial.

Figure 1-2. New Projects in the Nios II Perspective



Configuring the BSP

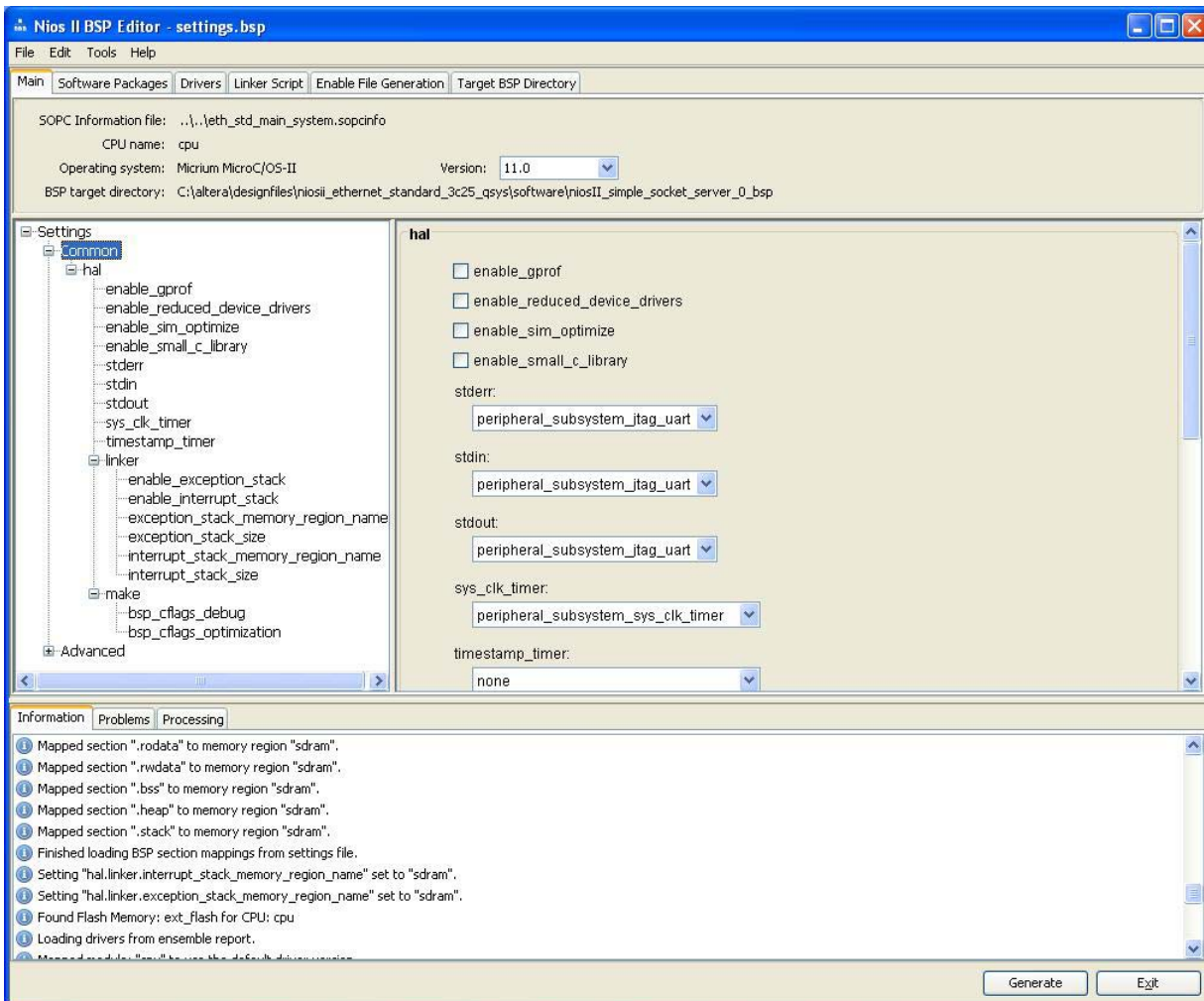
After you create a new BSP, you might want to customize its configuration (for example, defining `stdin`, `stdout`, `stderr`, and other parameters).

For more information, refer to the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

For this tutorial, you must configure the MicroC/OS-II RTOS kernel and NicheStack TCP/IP Stack software components. To do so, follow these steps:

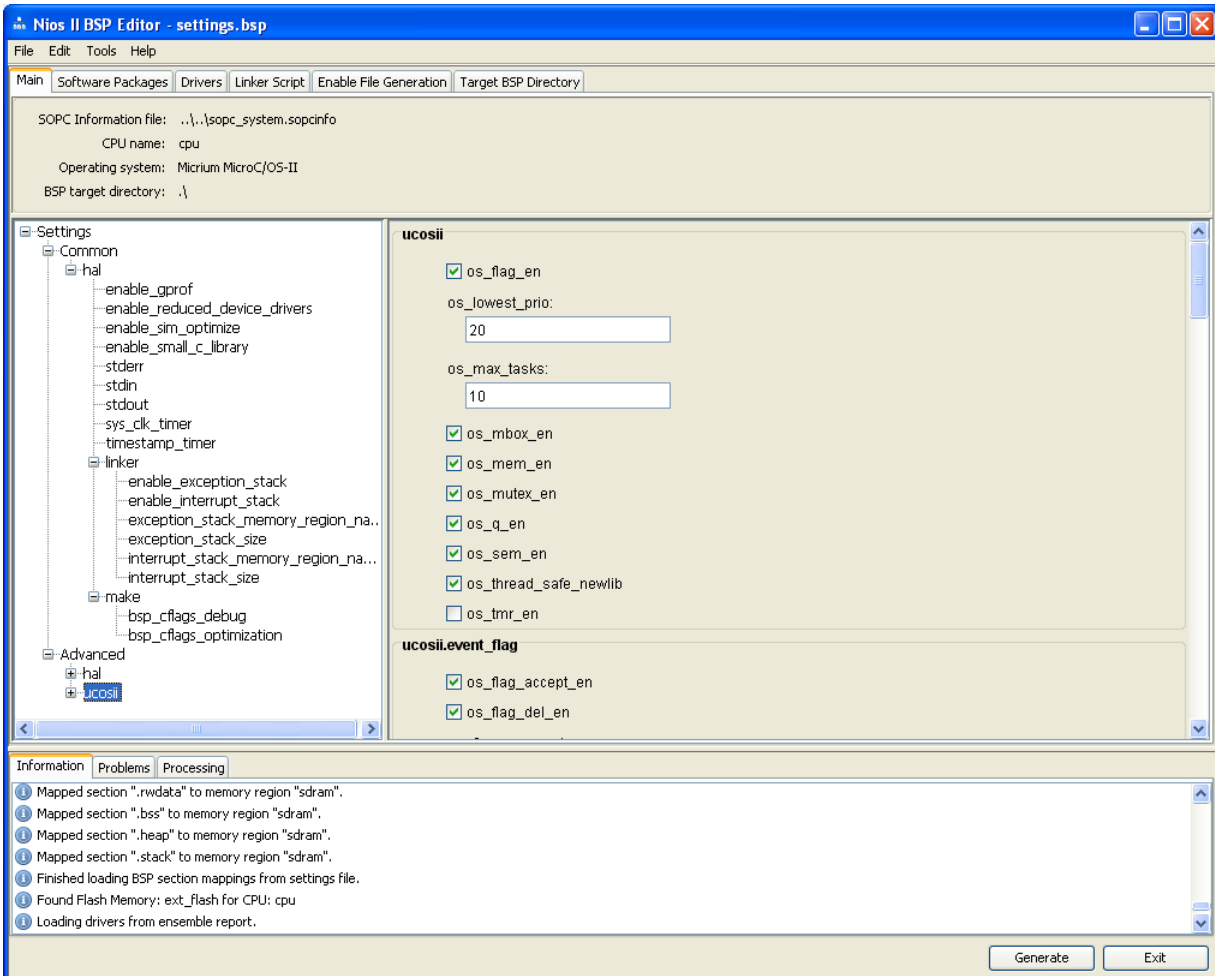
1. In the Project Explorer view, right-click the **niosII_simple_socket_server_0_bsp** project, point to **Nios II**, and click **BSP Editor**. The Nios II BSP Editor appears.
2. On the **Main** tab (**Settings** tab in v9.1 and earlier), expand **Settings** in the left pane, and click **Common**. Verify that the settings for the `stdout`, `stdin`, and `stderr` parameters are **peripheral_subsystem_jtag_uart**, as shown in Figure 1-3.

Figure 1-3. BSP Editor Main Tab




- On the **Main** tab (**Settings** tab in v9.1 and earlier), expand **Advanced** in the left pane, and click **ucosii**. Settings for the MicroC/OS-II RTOS appear, as shown in [Figure 1-4](#).

Figure 1-4. MicroC/OS-II RTOS Options



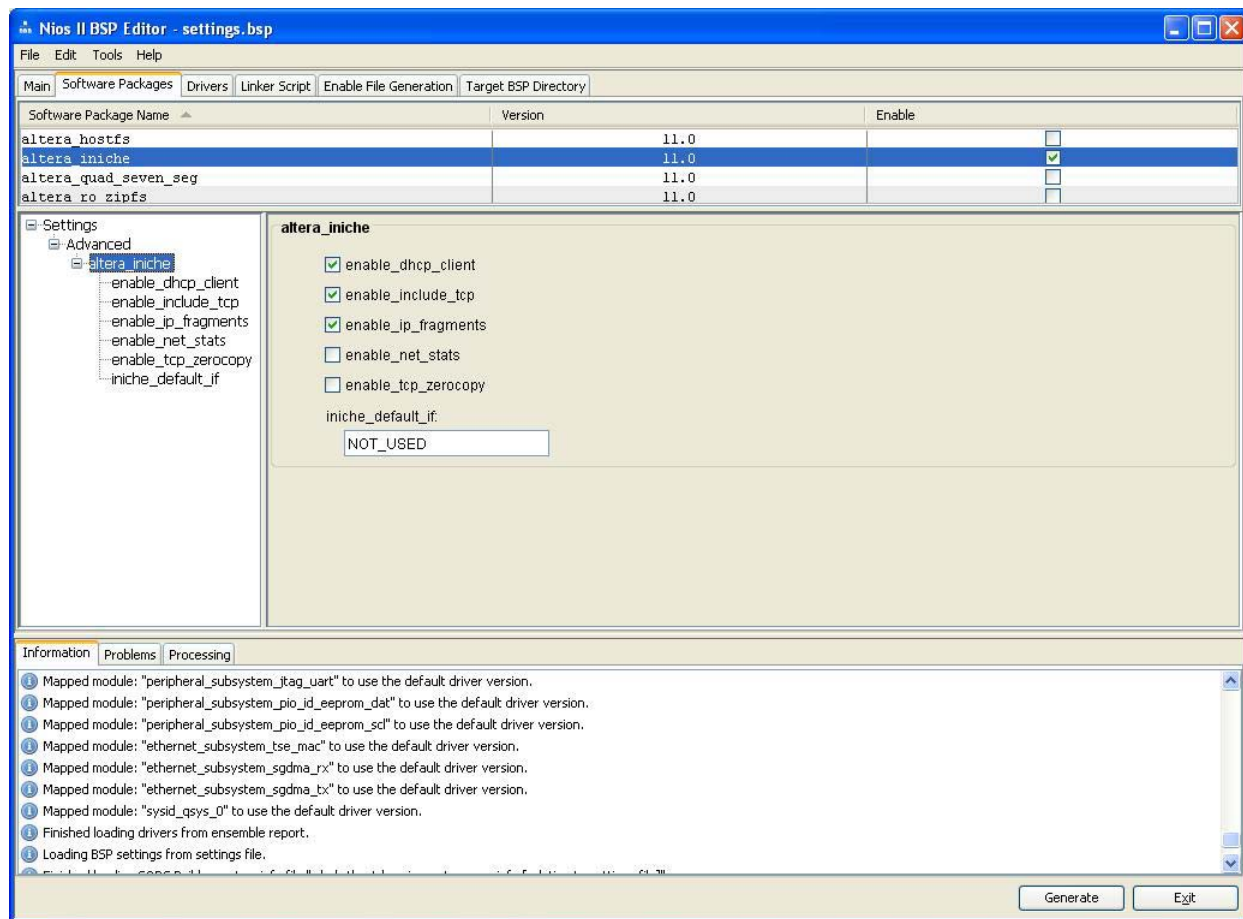
The MicroC/OS-II kernel is highly configurable. The options that you set in this dialog box determine which MicroC/OS-II options are included in the binary image. Examine the configurable options by clicking each of the options categories under **ucosii** in the left pane. For this tutorial, do not change any of the settings.

 Although this example software design does not use all the MicroC/OS-II system calls, the NicheStack TCP/IP Stack internally uses many more MicroC/OS-II system calls, more than the Nios II Simple Socket Server application itself uses. Do not disable any system calls unless you need to be very conservative with your code size requirements. You must reenale system calls that you try to disable if the link stage of the build fails with unresolved symbols.

For more information about the various MicroC/OS-II features, refer to the *MicroC/OS-II Real-Time Operating System* chapter in the *Nios II Software Developer's Handbook*.

- On the **Software Packages** tab, turn on **Enable** for the **altera_iniche** software package, as shown in Figure 1-5.

Figure 1-5. NicheStack TCP/IP Stack Options

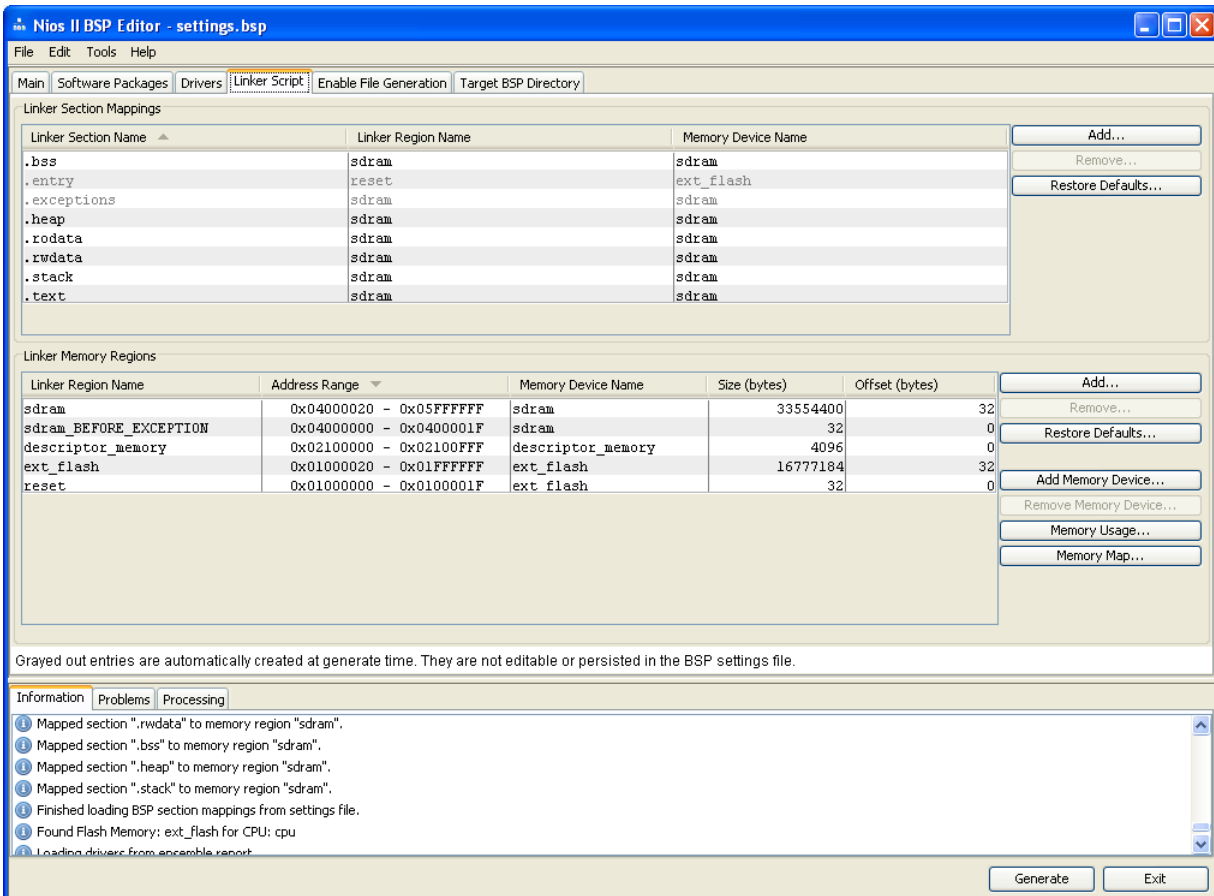


- If a DHCP server is available on your network, turn on **enable_dhcp_client**. If no DHCP server is available, turn off **enable_dhcp_client** and specify your IP addresses, gateway, and network mask in `<tutorial_files>\nichestack_tutorial\niosII_simple_socket_server.h`.

Take care when choosing your default IP and gateway addresses. Some secure router configurations block DHCP request packets on local subnetworks such as the 192.168.X.X subnetwork. If you do encounter problems, try using 0.0.0.0 as your default IP and gateway addresses.

- On the **Linker Script** tab, verify that the **Linker Region Name** is **sdram** for all enabled **Linker Section Names** in the table under **Linker Section Mappings**, as shown in Figure 1-6. If not, click each current **Linker Region Name** and select **sdram** from the list that appears.

Figure 1-6. BSP Editor Linker Script Tab



- Click **Generate**. When prompted to save your changes, click **Yes, Save**.
- Click **Exit** on the File menu to close the BSP Editor and return to the Nios II SBT for Eclipse.
- In BSP project, you must add **-DTSE_MY_SYSTEM** to your defined symbols. Right-click the **niosII_simple_socket_server_0_bsp** project and click **Properties**. The Nios II BSP Properties page appears. On the left pane, click Nios II BSP Properties. In the **Defined symbols** box, type **-DTSE_MY_SYSTEM**. Click **Apply** and **OK**.

Examining the Nios II Simple Socket Server Project Files

You have finished creating and configuring the `niosII_simple_socket_server_0` application and the associated BSP projects. Use the Project Explorer view, as shown in [Figure 1-2](#), to examine the project files in the `niosII_simple_socket_server_0` and `niosII_simple_socket_server_0_bsp` folders to understand more about the projects.

Building and Running the Nios II Simple Socket Server Project

This section guides you to run the design example on an Altera development board. This section also guides you to build the application, configure the development board with a hardware design, and download the executable software file to the FPGA on the board.

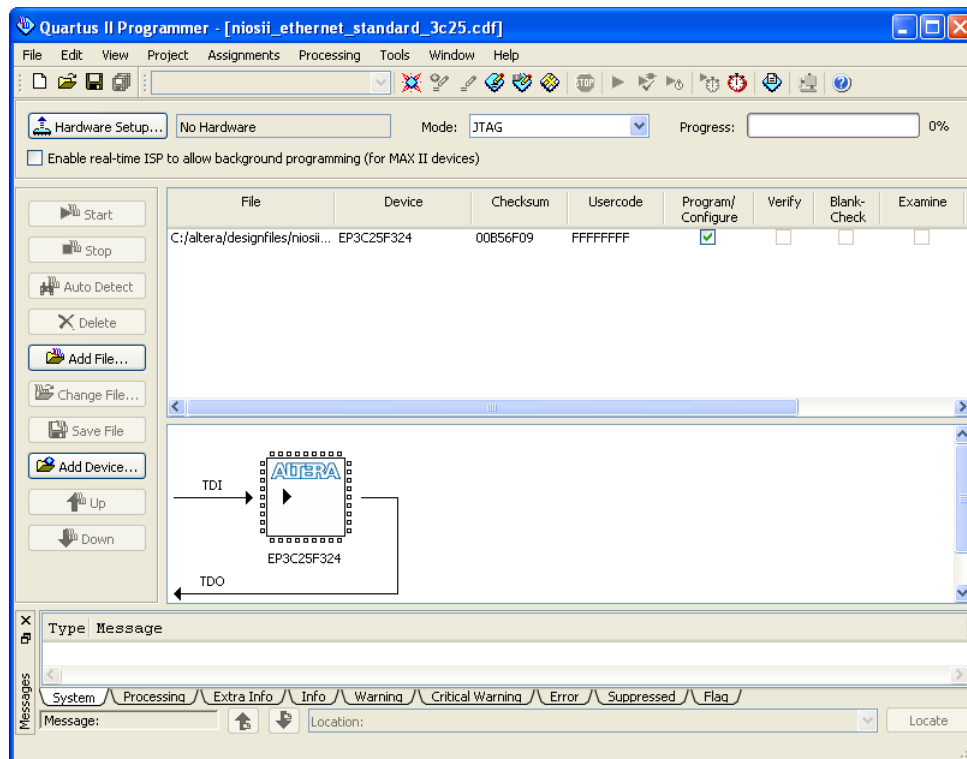


For more information about building and running programs with the Nios II SBT for Eclipse, refer to the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.


To build and run the application, follow these steps:

1. Configure the FPGA on the development board by performing the following steps:
 - a. On the Nios II menu, click **Quartus II Programmer**.
 - b. In the **Quartus II Programmer** dialog box, on the File menu, click **Open**.
 - c. Browse to and open the <tutorial_files>\niosii_ethernet_standard_<board>\niosii_ethernet_standard_<board>.sof SRAM Object File (.sof). Information for the file appears in the **Quartus II Programmer** dialog box.
 - d. Verify **Program/Configure** is on, as shown in Figure 1-7.

Figure 1-7. Quartus II Programmer Dialog Box



- e. Click **Start** to configure the FPGA on the development board.

 If **Start** is disabled or if the USB-Blaster cable is not listed in the **Hardware Setup** field, refer to the *Introduction to the Quartus II Software* manual for more information about the Quartus II Programmer.

- f. On the File menu, click **Exit** to close the Quartus II Programmer and return to the Nios II SBT for Eclipse. If you receive a message that asks if you want to save the changes to the **chain1.cdf** file, click **No**.
2. In the Nios II SBT for Eclipse, select the **niosII_simple_socket_server_0** project in the **Project Explorer** view.
3. On the Run menu, point to **Run As** and click **Nios II Hardware** to build the program, download it to the board, and run it.



If the **Run Configurations** dialog box appears, click the **Target Connection** tab. Then click **Refresh Connections** and **Apply** until a board connection establishes. After the board connection is established, click **Run**.

The build process takes several minutes. After the build process completes, the Nios II SBT for Eclipse downloads the executable program to your development board.



For additional information about using the Nios II SBT for Eclipse to build projects, set up run configurations, and download programs to the board, refer to the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

Interacting with the Nios II Simple Socket Server

After the program downloads to your development board, an LED starts blinking on your board. Table 1-1 identifies the LED that blinks on each development kit board.

Table 1-1. Blinking LED Identification

Kit	LED
ESDK, Cyclone III Edition	LED0
NEEK, Cyclone III Edition	LED1 (on back)
Stratix IV GX FPGA Development Kit	D23

The Nios II Console view displays a message with the default IP address as configured in `niosII_simple_socket_server.h`. If DHCP is enabled, the DHCP server-supplied IP address displays a message that indicates the DHCP client for the Ethernet interface acquires a DHCP IP address.

The message “Nios II Simple Socket Server starting up” displays when the NicheStack TCP/IP Stack is ready to accept commands.

After the NicheStack TCP/IP Stack is ready, you can start a telnet session to interact with the stack. To start a telnet session, follow these steps:

1. From your operating system, open a command shell.



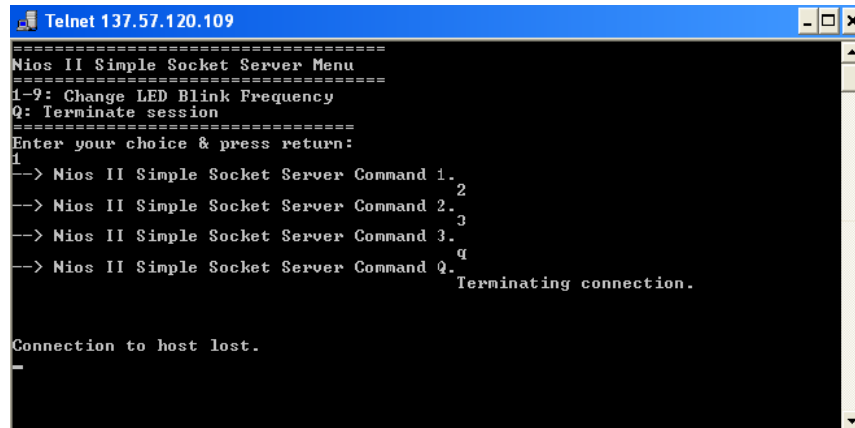
On Windows, you can also use **Run** on the Start menu.

2. Type the following command, specifying either the static IP address or the DHCP server-provided IP address:

```
telnet <IP address> 30 ↵
```

If the connection to port 30 on the development board is successful, the menu of available commands displays in a command window. When you enter commands at the command prompt, Ethernet sends the commands over the telnet connection to a task waiting on a socket for commands. The task responds to those commands by sending instructions to another task that manipulates the LED. [Figure 1-8](#) shows the Nios II Simple Socket Server menu, along with entered commands 1, 2, 3, and Q.

Figure 1-8. Interacting with the Nios II Simple Socket Server Via Telnet

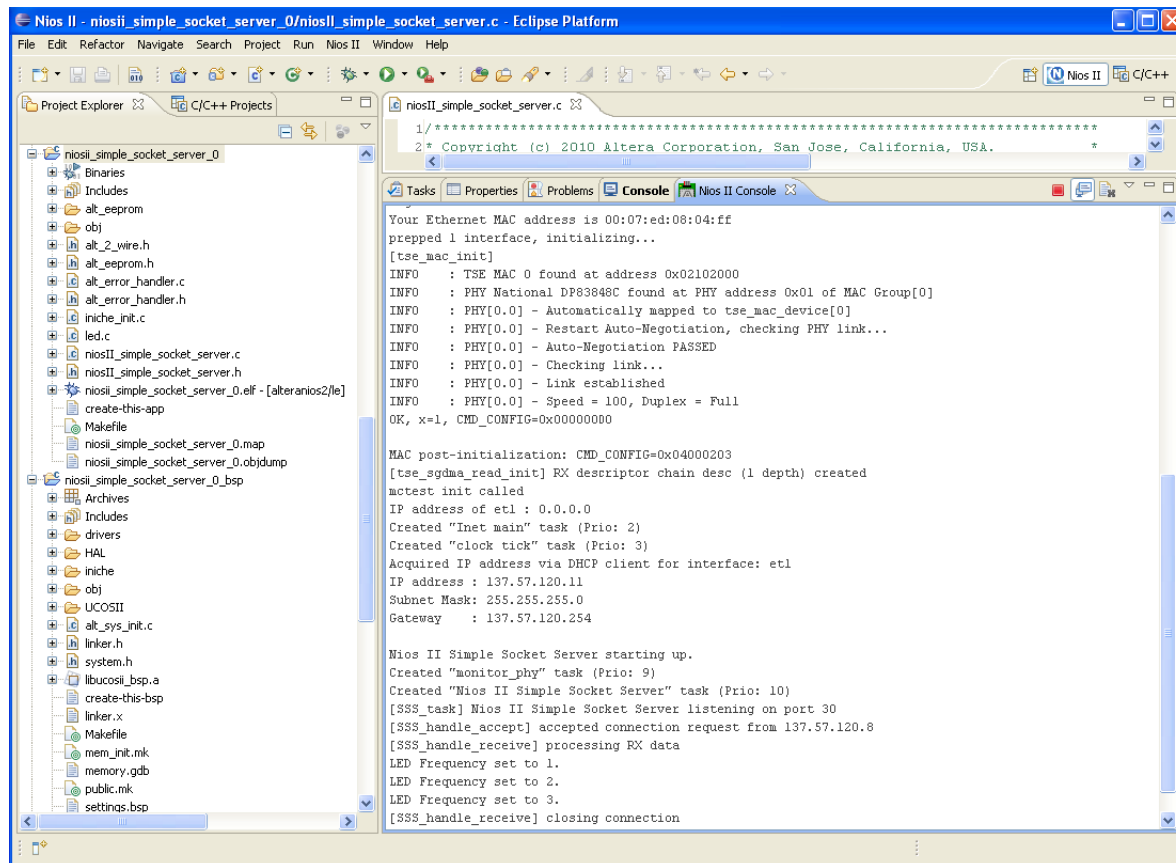


```
Telnet 137.57.120.109
=====
Nios II Simple Socket Server Menu
=====
1-9: Change LED Blink Frequency
Q: Terminate session
=====
Enter your choice & press return:
1
--> Nios II Simple Socket Server Command 1.
2
--> Nios II Simple Socket Server Command 2.
3
--> Nios II Simple Socket Server Command 3.
Q
--> Nios II Simple Socket Server Command Q.
Terminating connection.

Connection to host lost.
```

Figure 1-9 shows the corresponding output that appears in the Nios II Console view.

Figure 1-9. Nios II Console Output During Telnet Session



To test the functionality of the Nios II Simple Socket Server, enter the following commands in the telnet session:

1. Type `<n> ↵`, where `<n>` is a number from one through nine, to change the blink rate of the LED on your board.
2. Repeat step 1 several times, varying the number, to see the pulse rate change.
3. Type `Q ↵` to terminate the test. The socket connection on the development board terminates and the telnet command exits.

Nios II Simple Socket Server Overview

The following sections describe the Nios II Simple Socket Server:

- [“Software Naming Conventions” on page 1-16](#)—Identifies the naming convention used in the tutorial files.
- [“Software Architecture” on page 1-17](#)—Describes the architectural model of a Nios II software application and how it fits with the rest of the Nios II system software components.
- [“MicroC-OS/II Resources” on page 1-19](#)—Describes the tasks, queue, event flag, and semaphores that implement the Nios II Simple Socket Server software application.
- [“NicheStack TCP/IP Stack Initialization” on page 1-20](#)—Describes the required tasks and functions of the tutorial to establish and maintain the Ethernet TCP/IP socket connection.
- [“Nios II Simple Socket Server Implementation Details” on page 1-21](#)—Describes the structure that maintains the socket connection and the functions for each software component, including `main()`, MicroC/OS-II initialization, and the details of the SSS and LED software modules.

Software Naming Conventions

The Nios II Simple Socket Server uses capitalized acronym prefixes to identify public resources for each software module, and lowercase letters with underscores to indicate a private resource or function used internally to a software module. [Table 1-2](#) shows the software module acronym identifiers.

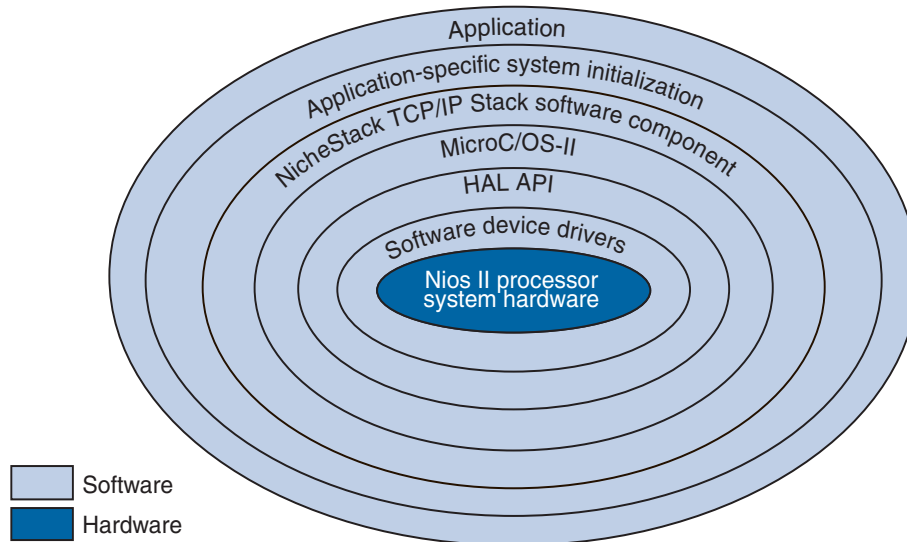
Table 1-2. Software Module Acronyms and Names

Acronym	Name
SSS	Nios II Simple Socket Server software module
LED	LED management software module
OS	MicroC/OS-II RTOS software component

Software Architecture

The onion diagram in [Figure 1-10](#) shows the architectural layers of a Nios II MicroC/OS-II software application.

Figure 1-10. Layered Software Model

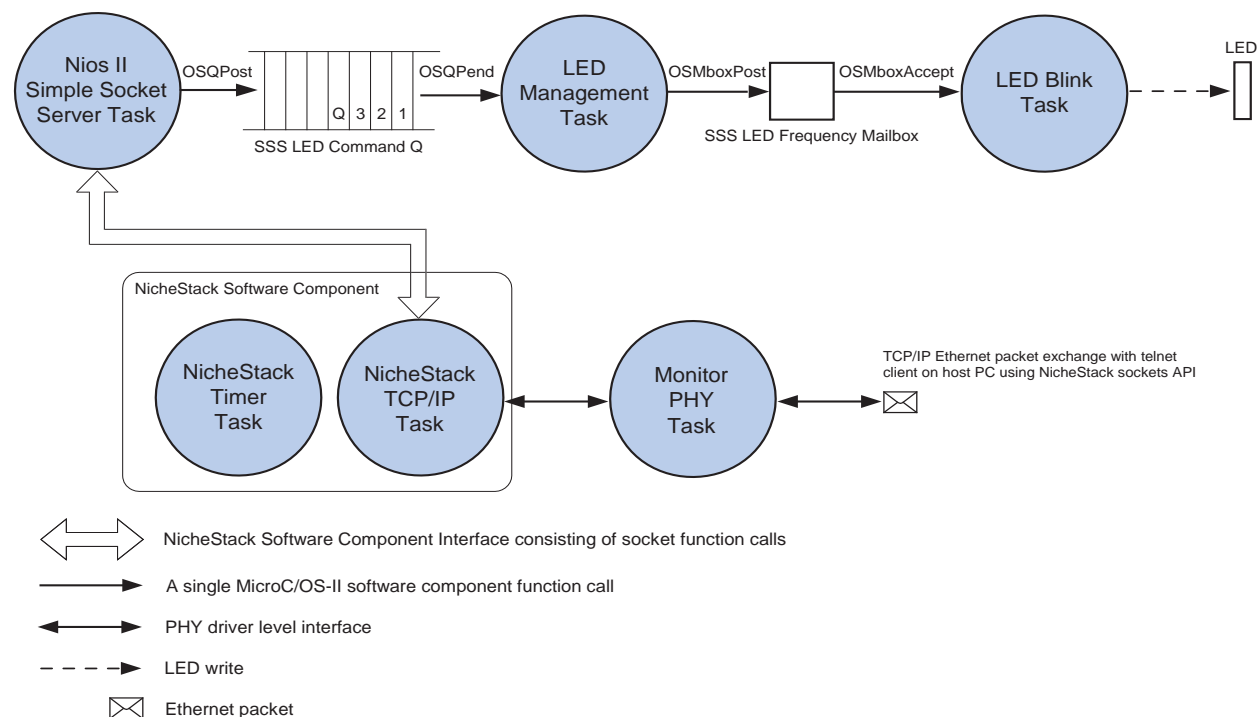


Each layer encapsulates the specific implementation details of that layer, abstracting the data for the next outer layer. The following list describes each layer:

- **Nios II processor system hardware**—The core of the onion diagram represents the Nios II softcore processor and hardware peripherals implemented in the FPGA.
- **Software device drivers**—The software device drivers layer contains the software functions that manipulate the Ethernet and hardware peripherals. These drivers know the physical details of the peripheral devices, abstracting those details from the outer layers.
- **HAL API**—The Altera Hardware Abstraction Layer (HAL) application programming interface (API) provides a standardized interface to the software device drivers, presenting a POSIX-like API to the outer layers.
- **MicroC/OS-II**—The MicroC/OS-II RTOS layer provides multitasking and inter-task communication services to the NicheStack TCP/IP Networking Stack and the Nios II Simple Socket Server.
- **NicheStack TCP/IP Stack software component**—The NicheStack TCP/IP Stack software component layer provides networking services to the application layer and application-specific system initialization layer via the sockets API.
- **Application-specific system initialization**—The application-specific system initialization layer includes the MicroC/OS-II and NicheStack TCP/IP Stack software component initialization functions invoked from `main()`, as well as creates all application tasks, and all the semaphores, queue, and event flag RTOS inter-task communication resources.
- **Application**—The outermost application layer contains the Nios II Simple Socket Server task and LED management tasks.

Figure 1-11 shows the structure of the design example described in “Introduction” on page 1-1. The sections following the figure describe the tasks in the figure.

Figure 1-11. Nios II Simple Socket Server Data Flow Diagram



The figure shows the state of the system after initialization. When the NicheStack TCP/IP Stack software component receives an Ethernet packet that contains an LED command sent from a telnet client program, the NicheStack TCP/IP Stack processes the incoming Ethernet packet with the TCP/IP protocol and presents the data packet to the socket server task using the sockets API. The LED management tasks then extract and post the LED command contained in the data packet to the LED command queue for processing.

MicroC-OS/II Resources

This section describes the tasks, queue, event flag, and semaphores that implement the Nios II Simple Socket Server application.

Tasks

Table 1-3 lists the MicroC/OS-II tasks that implement the Nios II Simple Socket Server application.

Table 1-3. MicroC/OS-II Tasks for the Nios II Simple Socket Server

Task	Description
SSSInitialTask()	Instantiates all the MicroC/OS-II resources. Initializes the NicheStack TCP/IP Stack and the Nios II Simple Socket Server example RTOS structures and tasks.
SSSNiosIISimpleSocketServerTask()	Manages the socket server connection, and calls relevant subroutines to manage the socket connection.
LEDManagementTask()	Manages LEDBlinkTask, driven by commands received via a MicroC/OS-II queue, named SSSLEDCmdQueue. The MicroC/OS-II mailbox, named SSSLEDFreqMailbox, passes blink rate values to the LED blink task.
LEDBlinkTask()	Blinks an LED on the development board, based on frequencies received via MicroC/OS-II mailbox.
SSSMonitorPhyTask()	Monitors the status of a single network PHY and maintains the network connection.

The application creates the tasks listed in Table 1-3. The NicheStack TCP/IP Networking Stack creates two additional software component layer tasks: a main task that operates the networking stack, and a time-keeping task that the main task uses. The application creates the NicheStack TCP/IP Stack main task (tk_netmain) in the netmain() function with a priority of TK_NETMAIN_TPRIO. The application creates the time-keeping task (tk_nettick) in the netmain() call with a priority level of TK_NETTICK_TPRIO. For more information about these tasks, and how to set their priorities and stack sizes, refer to “Important NicheStack TCP/IP Stack Concepts” on page 1-22.

Inter-Task Communication Resources

The following global handles (or pointers) create and manipulate your MicroC/OS-II inter-task communication resources. All the resources begin with SSS, indicating a public resource provided by the Nios II Simple Socket Server that is shared between software modules. The SSSCreateOSDataStructs function, invoked from SSSInitialTask() declares and creates these resources in **niosII_simple_socket_server.c**.

■ SSSLEDCommandQ

SSSLEDCommandQ is a MicroC/OS-II message queue that sends commands from the simple socket server task to the Altera development board LED control task, LEDManagementTask().

■ SSSLEDFreqMailbox

SSSLEDFreqMailbox is the handle to the MicroC/OS-II LED pulse rate mailbox. The mailbox passes an LED blink rate between the LED control task, named LEDManagementTask(), and the LED task responsible for blinking the development board LED, named LEDBlinkTask(). The LEDManagementTask() passes the pulse rate in response to a command sent from SSSNiosIISimpleSocketServerTask() when the “blink the LED” command comes over the TCP/IP socket.

NicheStack TCP/IP Stack Initialization

As described in the “NicheStack TCP/IP Stack Tasks” and “Initializing the Stack” sections of the *Ethernet and the NicheStack TCP/IP Stack – Nios II Edition* chapter of the *Nios II Software Developer’s Handbook*, the NicheStack TCP/IP Stack must be initialized from the Nios II Simple Socket Server application code by calling the following NicheStack functions:

- alt_iniche_init(), called from SSSInitialTask() in **iniche_init.c**
- netmain(), called from SSSInitialTask() in **iniche_init.c**

You must provide three NicheStack functions, get_mac_addr(), get_board_mac_addr() and get_ip_addr(). For this tutorial, these functions are available in **iniche_init.c**.



If get_board_mac_addr() is unable to find valid MAC address, refer to your board's user guide for instructions on restoring the MAC address, or hard code the MAC address in the get_mac_addr() function.

An initialization task named SSSInitialTask() calls the alt_iniche_init() and netmain() initialization functions in the proper sequence, and then waits until the NicheStack TCP/IP Stack becomes fully operational (by waiting for the global variable iniche_net_ready to be set to TRUE) before creating the application level task SSSNiosIISimpleSocketServerTask().

SSSNiosIISimpleSocketServerTask() is defined in **niosII_simple_socket_server.c** and created with priority SSS_NIOS_II_SIMPLE_SOCKET_SERVER_TASK_PRIORITY.



You can use the task SSSInitialTask() in your own MicroC/OS-II and NicheStack TCP/IP Stack networking application.

Nios II Simple Socket Server Implementation Details

This section provides details about the simple socket server structures, tasks, and functions.

The Nios II Simple Socket Server application uses following structure to manage each single socket connection:

```
typedef struct SSS_SOCKET {  
    enum { READY, COMPLETE, CLOSE } state;  
    int fd;  
    int close;  
    INT8U rx_buffer[SSS_RX_BUF_SIZE]; /* circular buffer */  
    INT8U *rx_rd_pos; /* position we've read up to */  
    INT8U *rx_wr_pos; /* position we've written up to */  
} SSSConn;
```

The application's `main()` function (located in `iniche_init.c`) performs the following actions:

- Calls `OSTimeSet()`
- Calls `OSTaskCreateExt` which in turn calls `SSSInitialTask()`
- Calls `alt_uCOSIIErrorHandler()`
- Calls `OSStart()` to begin multithreading

The Micrium's MicroC/OS-II examples suggest using a single task to initialize the rest of the application. This technique ensures that stack checking initializes enabled features correctly. In this tutorial, the `SSSInitialTask()` task (located in `iniche_init.c`) initializes the NicheStack TCP/IP Stack software, initializes the operating system data structures, and starts any user-defined networking tasks and regular tasks. The `SSSInitialTask()` task performs the following specific actions:

- Calls `alt_iniche_init()` to perform pre-initialization of the NicheStack Networking Stack
- Calls `netmain()` to initialize and start the NicheStack Networking Stack
- Instantiates `ssstask()` and `sssphytask()` (via `TK_NEWTASK`) to start the Nios II Simple Socket Server networking task
- Calls `SSSCreateOSDataStructs()` to create data structures (`SSSLEDCommandQ` and `SSSLEDFreqMailbox` RTOS resources) for the Nios II Simple Socket Server application
- Calls `SSSCreateTasks()` to create non-NicheStack TCP/IP Stack dependent tasks, including the LED tasks
- Calls `OSTaskDel()` to delete itself as a task

The `SSSNiosIISimpleSocketServerTask()` task (located in `niosII_simple_socket_server.c`) performs the following actions:

- Creates a socket to serve a TCP/IP connection, binds to the socket, and listens for TCP/IP connection requests from a client.
- Calls `SSS_handle_accept()` for an incoming TCP/IP connection.

- Calls `SSS_handle_receive()` to serve the TCP/IP connection. If you require multiple TCP/IP connections, you can modify this task to create other tasks that handle each individual TCP/IP connection.
- Calls `SSS_reset_connection()`, `SSS_send_menu()`, and `SSS_exec_command()`.
- When the `SSSNiosIISimpleSocketServerTask()` task receives data packets, the task extracts, and passes the LED commands to `LEDManagementTask()` via the `SSSLEDCommandQ`.

The following list describes the LED tasks (located in **leds.c**):

- The `LEDManagementTask()` task consumes and processes LED commands received on the `SSSLEDCommandQ`. The task reads the `SSSLEDCommandQ` for an incoming message command from `SSSNiosIISimpleSocketServerTask()`, converts the command to an LED pulse rate, and posts the pulse rate to the `SSSLEDFreqMailbox` mailbox for the `LEDBlinkTask()` task.
- `LEDBlinkTask()` blinks the LED based on the pulse rate the `LEDManagementTask()` task sets in the MicroC/OS-II `SSSLEDFreqMailbox` mailbox.

Important NicheStack TCP/IP Stack Concepts

The following topics could have a significant impact on your design.

Error Handling

A suite of error-handling functions defined in `alt_error_handler()` check error handling of the Nios II Simple Socket Server application, NicheStack TCP/IP Stack, and MicroC/OS-II system call error-codes. All system, socket, and application calls check for error conditions whenever an error could exist.

NicheStack TCP/IP Stack Default Task Creation

The NicheStack TCP/IP Stack creates one or more system level tasks during system initialization, when you call the `netmain()` function. Users have complete control over these system level tasks through a global configuration file named **ippport.h**, located in the directory structure for the BSP project, in the **iniche/src/h/nios2** folder.

You can edit the `#define` statements in **ippport.h** to configure the following options for the NicheStack TCP/IP Stack:

- **Module Inclusion**—Identifies which built-in NicheStack modules should be started
- **Module Priority**—Identifies what MicroC/OS-II priority the module task should use
- **Module Stack Size**—Identifies what MicroC/OS-II stack size the module should use



For more information about other NicheStack TCP/IP Stack options that can be enabled at run-time, refer to the NicheStack TCP/IP Stack documentation in **NicheStackRef.zip** located in the *<Nios II EDS install path>/components/altera_iniche/UCOSII/31src* directory.

In the Nios II Simple Socket Server design example, only the minimum required NicheStack TCP/IP Stack tasks have been configured to run. These tasks are as follows:

- `tk_netmain`—Initializes the stack, including networking interfaces
- `tk_nettick`—A time management task that the networking stack uses

For more information about these NicheStack TCP/IP Stack tasks, refer to “[Task Priorities in the Nios II Simple Socket Server Design](#)” on page 1-25.

Creating Tasks that Use the NicheStack TCP/IP Stack Sockets Interface

You must use the function call `TK_NEWTASK` to create any tasks that use the NicheStack networking services. You must create tasks that do not use networking services with the MicroC/OS-II function `OSTaskCreate()`.

The NicheStack Networking Stack uses the `TK_NEWTASK` (defined in `osportco.c`) function to launch MicroC/OS-II tasks that use the networking services. `TK_NEWTASK` accepts a single argument, `struct inet_taskinfo * nettask` (defined in `osport.h`), which specifies the task name, the MicroC/OS-II thread priority, and the stack size. You can locate these files in the `<Nios II EDS install path>/components/altera_iniche/UCOSII/src/nios2` directory. The `struct inet_taskinfo` structure is defined as follows:

```
struct inet_taskinfo {  
    TK_OBJECT_PTR(tk_ptr); /* pointer to static task object */  
    char * name; /* name of task */  
    TK_ENTRY_PTR(entry); /* pointer to code that starts task */  
    int priority; /* MicroC/OS-II priority of the task */  
    int stacksize; /* size (bytes) of task's stack */  
    char* stackbase; /* base of task's stack */  
};
```

For every networking task you create in your application, you must declare a local `struct inet_taskinfo` structure with the elements defined. These elements are listed in the following bullets, along with a brief explanation of their function:

- `TK_OBJECT_PTR(tk_ptr)`—A pointer to a static task object, defined for a given task via the `TK_OBJECT` macro. The NicheStack Networking Stack makes use of the `tk_ptr` element during the operation. After declaring the variable name via the `TK_OBJECT` and populating the `TK_OBJECT_PTR(tk_ptr)`, you do not need to do anything more.
- `char * name`—This element contains a character string that corresponds to the name of the task. You can set it with any character string you choose.
- `TK_ENTRY_PTR(entry)`—This element corresponds to the entry point or defined function name of the task that you want to run.
- `int priority`—The MicroC/OS-II priority level for the task.
- `int stacksize`—The MicroC/OS-II stack size for the task.
- `char* stackbase`—This element in the structure is used by the NicheStack software. You must not change the element in the structure.

In addition to declaring the struct `inet_taskinfo` structure, you must invoke two macro definitions: `TK_OBJECT` and `TK_ENTRY`. These macros have the following uses:

- `TK_OBJECT(name)`—Creates the static task object named `name`, which is used by NicheStack during operation. The static task object is also set in `TK_OBJECT_PTR(tk_ptr)`. A NicheStack naming convention for the `name` parameter is to set it to the string “to_”, followed by the declared name of the struct `inet_taskinfo` instance.
- `TK_ENTRY(name)`—Used to create a declaration of the task’s entry point, or function name. The `name` parameter is identical to the function name you specified for the task that you want to create, which must have the form `void name (void)`. The `name` parameter sets the `TK_ENTRY_PTR(entry)` macro.

To create your own application tasks that use the services offered by the NicheStack TCP/IP Stack, follow these steps:

1. **Invoke Task Macros**—Include the `TK_OBJECT` and `TK_ENTRY` macros, with information about your task.
2. **Define Task Parameters**—Define your task application by filling in a local `inet_taskinfo` structure in your code.
3. **Wait for Stack Initialization**—Before launching your task, wait until the external variable `iniche_net_ready` is set to `TRUE`. This variable sets to `FALSE` at run time and changes to `TRUE` when the NicheStack TCP/IP Networking Stack is operational.
4. **Launch Task**—Call `TK_NEWTASK` while passing in a pointer to the `inet_taskinfo` structure for your task.

Following is a code sample for creating your own application task:

```
// Declaration of SSSNiosIISimpleSocketServerTask
void SSSNiosIISimpleSocketServerTask(void) {
    // task specific code
}

// Creation of NicheStack networking task
TK_OBJECT(to_ssstask);
TK_ENTRY(SSSNiosIISimpleSocketServerTask);

struct inet_taskinfo ssstask = {
    &to_ssstask,
    "simple socket server",
    SSSNiosIISimpleSocketServerTask,
    TASK_PRIORITY,
    APP_STACK_SIZE,
};

while (!iniche_net_ready)
    TK_SLEEP(1);

/* Create the main simple socket server task. */
TK_NEWTASK(&ssstask);
```

Networking tasks can hand off large processing jobs that are independent of networking to other tasks. This task load segmentation has the advantage of increasing control over memory usage for task stacks, as well as greater control over prioritization of jobs.

Be careful not to overutilize job distribution among several tasks at the same time, for the following reasons:

- Additional tasks require additional CPU execution time to do task context-switching.
- Limited number of priorities. Each task must have its own unique priority in MicroC/OS-II, and you do not want to run out of task priorities.

Task Priorities in the Nios II Simple Socket Server Design

Task priorities in the application directly affect how the application runs, or if the task functions correctly at all. The MicroC/OS-II operating system uses a unique priority number scheme for running its tasks, in which tasks assigned a lower priority number are treated as higher priority tasks. Because the Altera version of the NicheStack TCP/IP Stack requires the use of the MicroC/OS-II RTOS for operation, all tasks run on the system must be assigned a unique priority number. For the Nios II Simple Socket Server demo application, all tasks have been assigned non-conflicting priorities. For your own application, however, you should verify that all tasks in your system are assigned unique priority numbers at run-time.

Table 1–4 lists the tasks that might be running in your system, and the mechanism for configuring the priority of these tasks.

Table 1–4. Simple Socket Server Tasks and Configuration Mechanisms

Task Type	Configuration Mechanism
MicroC/OS-II internal tasks	ucosii settings, located on the Main tab of the Nios II BSP Editor
NicheStack TCP/IP Stack internal tasks	ipport.h, located in the iniche/src/h/nios2 directory of your BSP project
Networking initialization task	iniche_init.c, located in the <i><tutorial_files></i> \nichestack_tutorial directory
User networking tasks (calls to TK_NEWTASK)	Created in the user application code
User non-networking tasks (calls to OSTaskCreate)	Created in the user application code
PHY monitoring task	Created in the user application code

The following sections discuss the priorities of the tasks in the Nios II Simple Socket Server design:

MicroC/OS-II Internal Tasks

The Nios II Simple Socket Server application has been configured to not use any MicroC/OS-II internal tasks.

NicheStack TCP/IP Stack Internal Tasks

TK_NETMAIN_TPrio, defined in ipport.h, sets the priority to a value of 2 for the main NicheStack TCP/IP Stack task, launched by netmain(). This task implements the core functionality of the NicheStack TCP/IP Stack. To maximize the TCP/IP packet-throughput rate, the priority of this task should be higher than application tasks that use the NicheStack TCP/IP Networking Stack.

TK_NETTICK_TPrio, defined in `ipport.h`, sets the priority to a value of 3 for the NicheStack TCP/IP Stack time-keeping task, launched by `netmain()`. The NicheStack TCP/IP Stack uses this task to keep track of time-based events in the networking stack. Altera recommends that you set the priority of this task to one priority level lower than `TK_NETMAIN_TPrio`.

Networking Initialization Task

`SSS_INITIAL_TASK_PRIORITY` is set to a value of 5 for the first task that MicroC/OS-II runs. This task creates the resources and all the other tasks before deleting itself. This task is given a high priority, not due to its high time-period rate or low latency requirement, but to create all the real-time operating system resources and tasks before the other tasks start using the resources.

User Networking Tasks

`SSS_NIOS_II_SIMPLE_SOCKET_SERVER_TASK_PRIORITY` is set to a value of 10, a priority that is lower than the consumer task `LEDManagementTask()`. The priority of this application task is set lower than all of the software components' system service tasks. This practice allows for the best overall scheduling latency, because the software component tasks are designed to operate for as short a period of time as possible.

User Non-Networking Tasks

`LED_MANAGEMENT_TASK_PRIORITY` is set to a value of 7. This task's function is to receive LED command messages from the `SSSNiosIISimpleSocketServerTask`.

`LED_BLINK_TASK_PRIORITY` is set to a value of 11. The priority of this application task is set lower than the rest of the tasks in the system because it requires very little of the Nios II processor's cycles to operate.

PHY Monitoring Task

`SSS_MONITOR_PHY_TASK_PRIORITY` is set to value 9. This task monitors the status of a single network PHY and acts to maintain the network connection.

Task Stack Size

Task stack space requirements depend on how the Nios II processor, HAL, RTOS, and individual software components are configured. A quick empirical check of the `Stk[]` array values at runtime, via the Nios II SBT for Eclipse memory window, is an easy way to examine the top of a task stack. Examination of a task's `Stk[]` array reveals differing values representing the used portion of the stack followed by multiple zeros where the stack has not yet reached. The number of zeros until the beginning of the next adjacent task stack shows how deep the stack has grown since the last system reset.

All tasks that make run-time library calls have space allocated from the top of the stack for the approximately 900-byte `_reent` structure. Each task has its own copy of the structure positioned on the task's stack. The size of this structure alone reduces the amount of available stack space.



For more information about the `_reent` structure, refer to the “The Newlib ANSI C Standard Library” and the “Implementing MicroC/OS-II Projects for the Nios II Processor” sections of the *MicroC/OS-II Real-Time Operating System* chapter of the *Nios II Software Developer’s Handbook*.

Where to Go Next

This example is easily expandable to handle multiple TCP connections on a single port. The `SSSNiosIISimpleSocketServerTask()` task could be modified to create separate socket connection instance tasks to handle multiple telnet connections.

There are many uses for an Ethernet connection in an embedded system. A connection to the Internet can allow the addition of many powerful features for any embedded product, such as remote configurability using a web browser, or remote software upgrade for corrections or feature enhancements to a product already in the field.

Introduction

To complete this tutorial, you must have the Nios II SBT for Eclipse installed, and your Altera development board must be connected to a host PC on both the Ethernet and USB/JTAG ports.



For information about download cables and drivers, refer to the [Download Cables](#) page of the Altera website.

The Nios II Ethernet Standard hardware design examples for Altera development boards include the Ethernet device required by this NicheStack tutorial. The Ethernet device included in these design examples, along with the physical MAC/PHY on your Altera development boards, is the Altera Triple Speed Ethernet MAC peripheral. The Ethernet peripheral base address settings for the design examples are defined in `system.h`.

Network Connection

If you are using a DHCP server to assign IP addresses, connect your Altera development board to your Ethernet network.

If the Altera development board is connected directly to your PC with a crossover Ethernet cable, or a DHCP server is not available, specify the IP addresses manually in `niosII_simple_socket_server.h`.

The default IP addresses in `niosII_simple_socket_server.h` are set to all zeros so the DHCP server packets can pass through secure routers. If you are not using a DHCP server, specify valid static addresses, such as an IP address of 192.168.1.234, with a gateway of 192.168.1.1 and a subnet mask of 255.255.255.0.



Be sure to turn the `enable_dhcp_client` setting on or off accordingly on the **Software Packages** tab of the BSP Editor. For details, refer to “[Configuring the BSP](#)” on [page 1–7](#).

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
June 2011	3.0	Revised for Quartus II Software 11.0 release.
May 2010	2.0	Revised for Nios II Software Build Tools for Eclipse.
January 2007	1.0	Initial release.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General) (Software Licensing)	Email	nacomp@altera.com
	Email	authorization@altera.com








Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \qdesigns directory, d: drive, and chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Indicate document titles. For example, <i>AN 519: Stratix® IV Design Guidelines</i> .
<i>italic type</i>	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, <file name> and <project name>.pdf file.

Visual Cue	Meaning
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, <code>data1</code> , <code>tdi</code> , and <code>input</code> . The suffix <code>n</code> denotes an active-low signal. For example, <code>resetn</code> . Indicates command line commands and anything that must be typed exactly as it appears. For example, <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword <code>SUBDESIGN</code>), and logic function names (for example, <code>TRI</code>).
	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	A question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page on the Altera website, where you can sign up to receive update notifications for Altera documents.